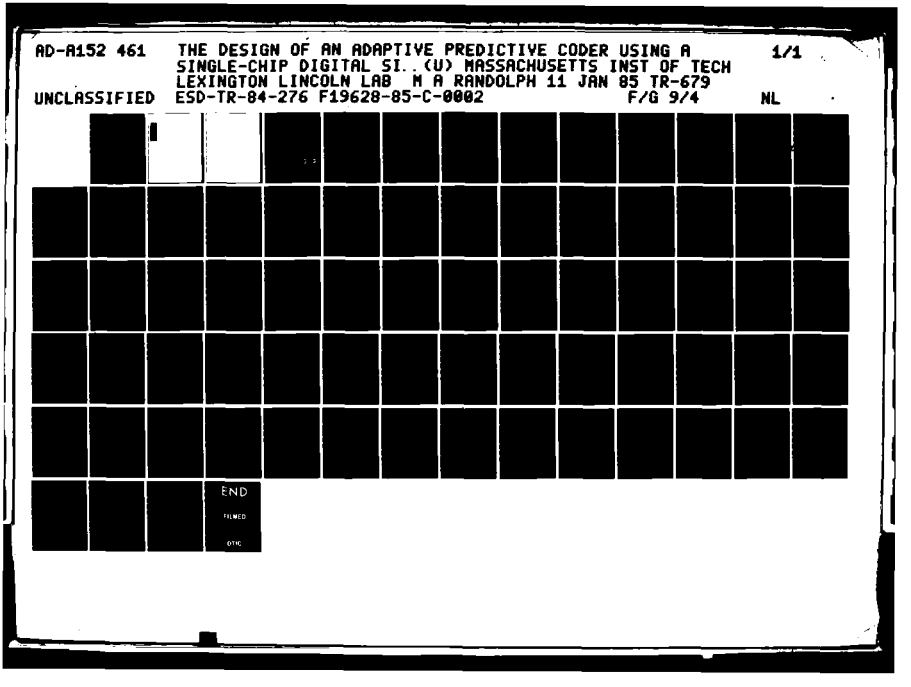
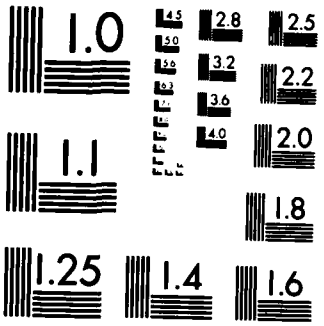

AD-A152 461 THE DESIGN OF AN ADAPTIVE PREDICTIVE CODER USING A 1/1
SINGLE-CHIP DIGITAL SI..(U) MASSACHUSETTS INST OF TECH
LEXINGTON LINCOLN LAB M A RANDOLPH 11 JAN 85 TR-679
UNCLASSIFIED ESD-TR-84-276 F19628-85-C-0002 F/G 9/4 NL
END
FILMED
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

THE DESIGN OF AN ADAPTIVE PREDICTIVE CODER USING A SINGLE-CHIP DIGITAL SIGNAL PROCESSOR

M.A. RANDOLPH

Group 24

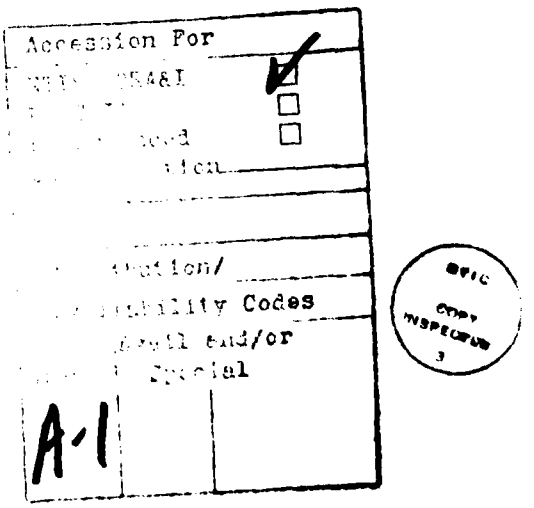
TECHNICAL REPORT 679

11 JANUARY 1985

LEXINGTON MASSACHUSETTS

ABSTRACT

A speech coding processor architecture design study has been performed in which the Texas Instruments TMS32010 has been selected from among three commercially available digital signal processing integrated circuits and evaluated in an implementation study of real-time Adaptive Predictive Coding (APC). The TMS32010 has been compared with the AT&T Bell Laboratories DSP I and Nippon Electric Co. μPD7720 and was found to be most suitable for a single chip implementation of APC. A preliminary system design based on the TMS32010 has been performed, and several of the hardware and software design issues are discussed. Particular attention was paid to the design of an external memory controller which permits rapid sequential access of external RAM. As a result, it has been determined that a compact hardware implementation of the APC algorithm is feasible based on the TMS32010.



CONTENTS

1. INTRODUCTION

Recently, several digital signal processing integrated circuits (DSPs) have become commercially available. These devices possess significant computational capability and permit a variety of speech processing algorithms to be implemented in compact, low power systems. In this report we summarize the results of a processor design study in which the Texas Instruments (TI) TMS32010, the Nippon Electric Co. (NEC) μPD7720, and the AT&T Bell Laboratories DSP I have been evaluated for the task of implementing real-time Adaptive Predictive Coding (APC). We have surveyed the architectural features of these three DSPs and have compared and contrasted their expected performance in implementing real-time APC.

Digital signal processors are typically benchmarked using some of the more common signal processing algorithms such as digital filters and FFTs. They are usually compared solely on the basis of the execution times of these computations. Unfortunately, we have found that in evaluating DSPs for real-time speech coding applications, these typical signal processing benchmarks do not provide us with complete information. For this reason, we have chosen to use an actual speech coding algorithm, real-time APC, as a benchmark. The decision to use APC was based on a number of factors. First, it is an algorithm of moderate to high complexity that requires a processor with considerable numerical processing capability. In addition, it requires a processor which can access an extensive amount of memory. It therefore provides a reasonable indication of the processing power of a particular digital signal processor. Secondly, it fits within the category of medium- to low-bit rate speech coding algorithms which we are currently

interested in implementing at Lincoln Laboratory. We postulate that a DSP's ability to implement APC is reasonable assurance that comparable algorithms could also be implemented on that DSP.

This report shall be organized as follows. In section 2, we describe pertinent aspects of the APC algorithm as they relate to the algorithm's implementation. In Section 3, we briefly review and compare the architectural features of the three digital signal processors that we have considered. In Section 4, we summarize a software/hardware implementation of the APC algorithm based on the TMS32010.

2. ADAPTIVE PREDICTIVE CODING

In the present section, we briefly outline the fundamentals of the APC algorithm. This discussion is intended to serve as a means of introducing our own terminology and notation. We have chosen not to develop the theory upon which the algorithm is based. For a more complete treatment of APC in terms of its theoretical aspects, we refer the reader to a report by Viswanathan, et al., [16], which is a comprehensive review of the theory and also describes several variations and improvements that have been made upon the APC algorithm. For the purpose of this study, we have considered the APC algorithm in its most basic form. This particular version of the APC algorithm is similar in structure to the original proposed by Atal and Schroeder [2]. A block diagram is shown in Fig. 1. Figures 1 (a) and (b) are the APC analyzer and synthesizer, respectively. In the analyzer, two predictors are employed for removing presumed redundancy in the input speech signal and are arranged in a feedback loop surrounding a one-bit quantizer. The predictor A(z) is a spectral predictor and is intended to

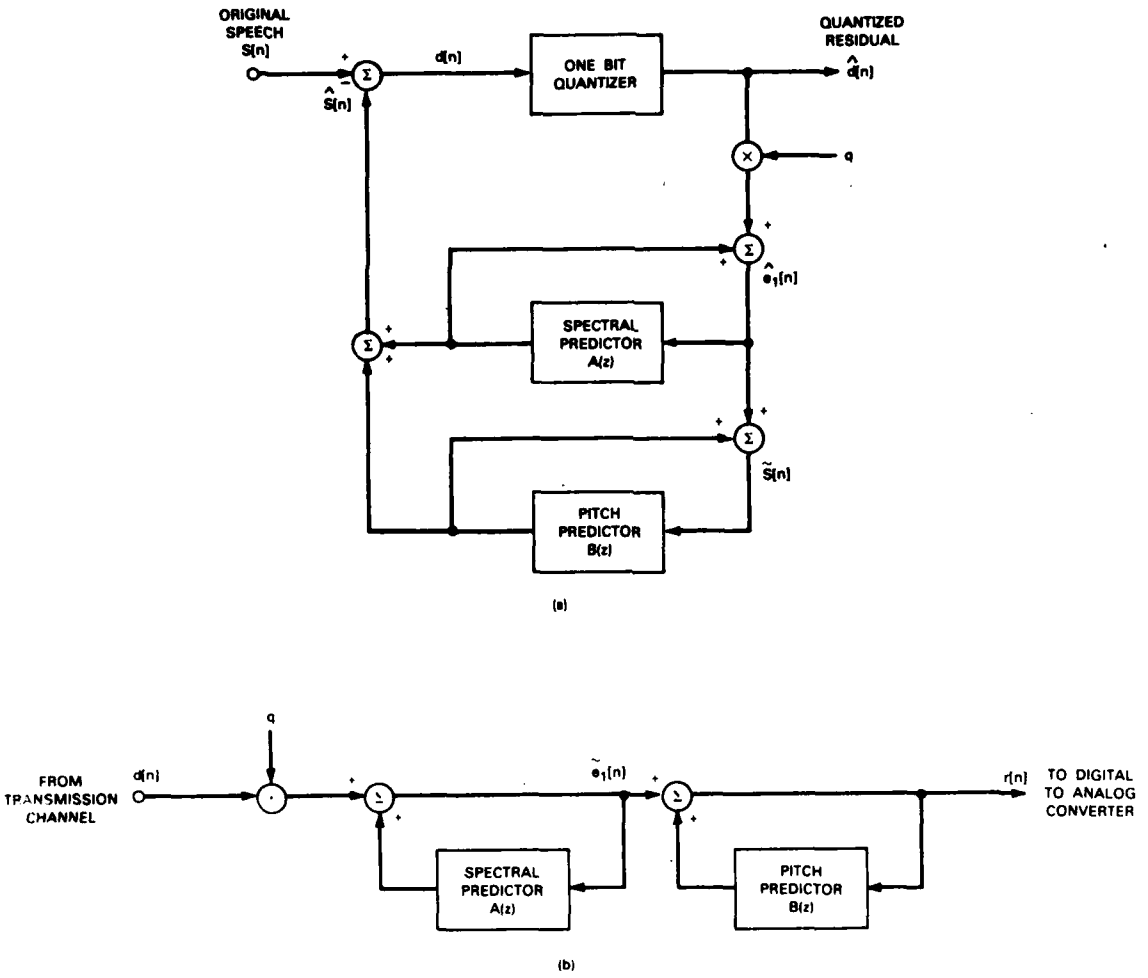


Fig. 1. Block diagrams of APC (a) analyzer and (b) synthesizer.

remove the redundancy in the speech signal which is due to its quasi-stationary spectral properties. The spectral predictor has the polynomial transfer function

$$A(z) = \sum_{i=1}^{P} a_i z^{-i} \tag{1}$$

where the coefficients a_i are the spectral predictor coefficients and the parameter P is the prediction order. The spectral predictor coefficients are obtained from linear prediction methods which will be outlined below. The prediction order, P, is usually a specification of the particular implementation and is typically equal to 4.

The second predictor, B(z), is the pitch predictor. It removes redundancy in the speech signal that is due to the quasi-periodicity of voiced sounds. The pitch predictor has the transfer function

$$B(z) = \alpha z^{-T} \tag{2}$$

where α is the pitch prediction coefficient and T is the estimated pitch period.

In the block diagram of Fig. 1, we show pitch prediction being followed by spectral prediction. At sample n, the predicted speech signal, $\hat{s}[n]$, is subtracted from the incoming speech signal, s[n], and the resulting residual, d[n], is quantized, coded and transmitted to the receiver. The quantized residual is also fed back within the analyzer loop. In the receiver, the spectral prediction signal is computed first and is added to the received residual before the pitch prediction signal. The pitch prediction signal is then added in, and the resultant synthesized

speech signal is passed to a digital to analog converter for the reconstruction of the analog speech signal.

The one-bit quantized residual fed back within the analyzer and the decoded residual $\tilde{d}[n]$ in the receiver are both unit variance signals and are scaled by a multiplicative factor q. The factor q is an estimate of the standard deviation of d[n] and forces the quantized residual signals in both the analyzer and the receiver to be equal to the original residual, d[n], with the addition of quantization noise.

The methods used to compute the APC side parameters α, T, a_i and q are understood by viewing the prediction operations in the time domain. Removal of the pitch redundancy in the input speech signal can be written as the difference equation

$$e_1[n] = s[n] - \alpha s[n-T] \tag{3}$$

in which the signal $e_1[n]$ is referred to as the first residual. If the signal s[n] were exactly periodic, and if T were computed without error, the coefficient α would equal one, and the first residual would be identically equal to zero. However, since speech is never exactly periodic and since the pitch estimation process employed in APC does produce errors, the first residual can never be identically zero in any practical sense. Thus, once the pitch period has been determined, α is estimated in a manner which minimizes the mean square energy in the first residual. This results in the following expression for α, the normalized correlation coefficient

$$\alpha = \frac{\sum_{n=\emptyset}^{N-1} s[n]s[n-T]}{\sum_{n=\emptyset}^{N-1} s[n-T]s[n-T]} \tag{4}$$

Computing the spectral predictor is actually the linear prediction analysis problem. In this context, instead of performing the analysis on the speech signal, linear predictive analysis is performed on the first residual, $e_1[n]$. Linear prediction methods have been examined thoroughly in the literature and several solutions to the resulting normal equations have been proposed (see for example [10], [11], and [12] for solution methods). In the implementation of APC given in this report, we have used the autocorrelation method of solving the linear prediction normal equations. Standard algorithms, such as Levinson's recursion and the LeRoux-Gueguen recursion, which allow the predictor coefficients to be obtained from the first P+1 autocorrelation values, have both been considered.

The parameter q, as we have mentioned above, is an estimate of the standard deviation of the residual d[n] and is used to scale the standard deviation of the quantized residual $\hat{d}[n]$ in the analyzer and $\tilde{d}[n]$ in the receiver to the same level as d[n]. Thus, when the quantized residual is used in the feedback loop in the analyzer, the reconstructed first residual $\hat{e}_1[n]$ and the reconstructed speech $\hat{s}[n]$ become equal to $\tilde{e}_1[n]$ and r[n], the first residual and speech signals in the receiver in the absence of transmission errors. Note also that all of these signals will have been degraded by the identical quantization noise introduced at the analyzer.

The pitch period, T, can be estimated using any number of methods in APC. However, practical considerations permit only simple methods to be employed and pitch errors will typically be introduced. This is not a major disadvantage, however, since pitch errors do not severely degrade the

:ransmission rates, we begin this summary of the algorithm's implementation)y describing, in closer detail, the specifications of the APC algorithm :hat we propose to implement. In Section 4.2 we give results of a critical loop timing study, and we describe a control strategy for fitting together :he various software components of the APC algorithm. We have concluded from this programming exercise, in which the critical loops of the APC algorithm were coded in the actual TMS32010 instruction set, that the most important consideration in designing the software for APC is the fashion in which data is stored in memory. In section 4.3 we describe one possible memory allocation scheme. We close our discussion of the APC implementation by describing the hardware requirements. We have concluded from performing a preliminary hardware design that the majority of the hardware design effort must be directed towards providing a high speed interface with memory external to the TMS32010, and to developing a method of communicating with several external input/output devices under interrupt control. The details of this preliminary hardware design are summarized in Section 4.4.

4.1 Algorithm Specifications

In Section 2 we outlined the structure of the APC algorithm. The structure described in Section 2 will support a range of data transmission and speech sampling rates. The version of the APC algorithm chosen for this study is intended to operate at a transmission rate of 9.6 Kbps and at a sampling rate of 8000 samples/sec. The average frame duration is intended to be 20 msec which corresponds to an analysis frame size of 160[1]

[1]Note that the frame duration is measured with respect to the transmission and receiver modem clocks which are asynchronous to the sampling rate clock. Therefore, the analysis frame may deviate slightly from the 160 sample nominal size.

implementation should require a minimal amount of hardware, the TMS32010 seems to be the best alternative among the three. Using the AMDF pitch estimation computation as a comparison task, the Bell Labs DSP would require the most extensive amount of external support hardware followed by the NEC μPD7720 requiring an external memory controller and a control microcomputer, and, lastly, the TMS32010 requiring just an external memory controller.

In this section we have made comparisons of these DSPs based primarily on their memory accessing capabilities. Another important distinguishing feature which allows these DSPs to be compared is their capability for providing foreground/background multi-tasking of computations. In this respect, we require a DSP to have the ability to handle interrupts. As far as satisfying this particular requirement, the Bell Labs DSP does not support interrupts while the NEC μPD7720 and the TMS32010 do.

Based on the issues discussed in this section the TMS32010 is the processor of choice among the three DSPs that we have evaluated for the task of implementing APC. Although complete evaluations based on speech coding algorithms other than APC have not been carried out, it is reasonable to assert that the TMS32010 would be most appropriate for a variety of other moderate complexity speech coding algorithms as well.

4. APC PROCESSOR DESIGN

For the remainder of this report, we shall summarize the results of this APC processor design study by briefly describing one possible hardware/software implementation of real-time Adaptive Predictive Coding using the Texas Instruments TMS32010. Because the basic APC structure described in Section 2 will support a variety of speech sampling and data

TMS32010 transfers data over its parallel I/O ports. Although both modes of external memory access physically involve the data bus, the differences between these two modes are important. Mode I memory transfers are effected by executing a three machine cycle **TBLR** or **TBLW** instruction in the TMS32010. When executing these instructions, the contents of the CPU accumulator are taken and used directly as the memory address. In mode II memory transfers, a two machine cycle **IN** or **OUT** instruction is executed. For these instructions the least significant three bits of the address bus contain a port address which can be decoded externally to select one of eight devices that are to send or receive data from the TMS32010 over the data bus. When the I/O ports are used for memory access, ROM/RAM addresses must be provided externally. The trade-off between using these two modes of memory I/O is one made between hardware and software efficiencies. Although the **TBLR** and **TBLW** instructions nominally require three machine cycles to execute, extra machine cycles are typically required for saving and restoring the contents of the accumulator which are involved in the ongoing computation. As a result, instead of three machine cycles being required for memory I/O, the amount of time often turns out to be on the order of seven to eight machine cycles. On the other hand, memory I/O involving the data ports is guaranteed to require no more than two machine cycles. The disadvantage in the case of mode II is that external hardware is needed for generating the required memory addresses.

3.4 Discussion and Summary

With ample external support hardware, any of these DSP integrated circuits could be used to implement real-time APC. However, given that the

3.3 Texas Instruments TMS32010

The Texas Instruments TMS32010 effectively combines the high numerical processing power of the NEC μPD7720 SPI with the control, data manipulation, and storage capabilities previously found only in general purpose microprocessors. A full description of the Texas Instruments TMS32010 architecture is given in [15]. We have highlighted some of its features here. They include:

-a 1500 x 16-bit internal ROM,

-an external ROM/RAM memory address space of up to 4K words,

-a 144 x 16-bit internal RAM,

-eight 16-bit parallel I/O ports,

-a 200 nsec 16 x 16-bit parallel multiplier with a 32-bit ALU/accumulator,

-a 200 nsec machine cycle time.

The most important advantage that the TMS32010 offers over the Bell Laboratories DSP and the NEC μPD7720 SPI is its 12 bits of external memory address space. A 16-bit word can be accessed from external ROM/RAM in two to three machine cycle instructions. We have found that based on this relatively short memory access time, the sum of the execution times of all of the critical loops of the APC algorithm is less than a 20 msec frame duration (see Section 4.2 below). Therefore, the APC algorithm could most likely run in real-time in a single chip, TMS32010-based system.

The TMS32010 provides two modes of accessing off-chip memory. In mode I, the TMS32010 generates the necessary memory addresses internally, and data is transferred over the 16-bit data bus. In mode II, the

manipulating these internal memory pointers. This programming inconvenience makes stream processing of data in the APC algorithm preferable over the use of block processing methods, which would buffer data needed for parameter computation in internal RAM.

Assuming data is to be processed in a stream fashion, we were able to approximate the execution times of some of the critical loops of the APC algorithm. Assuming external control of the system data paths, as shown in Fig. 2, approximately 4 μsec per 16-bit word are required to exchange data with external memory [8]. If one adds these numbers up, the approximated execution time of the AMDF pitch estimation algorithm is in excess of an analysis frame duration. However, the execution times of the remaining APC critical loops are each shorter than the assumed frame duration of 20 msec, thereby making a NEC μPD7720 based implementation of real-time APC feasible if an alternative pitch estimation algorithm to the AMDF method is used. These results are particularly encouraging, since it has already been demonstrated in the Lincoln Laboratory compact LPC vocoder that the more sophisticated, but less memory intensive, Gold pitch estimation algorithm can be programmed to run in the NEC μPD7720 in real time.

From these observations it seems that a real-time implementation of APC based on the NEC μPD7720 SPI is feasible. The architecture of such a system would most likely resemble the one shown in Fig. 2. Further determination of the specific hardware and software complexity, such as the exact number of NEC μPD7720 SPIs that would be required, has not been undertaken and, of course, would be the next step. Instead, our attention has been directed towards determining the feasibility of the Texas Instruments TMS32010.

external RAM with a separate DMA controller. Both of these approaches have been included in the architecture shown in Fig. 2. The DMA controller in the system architecture is used to handle block (and stream) data transfers between the SPIs and the external RAM, and the control microcomputer is used to control the data flow between the SPIs.

In addition to the large amount of memory required, the computational requirements of real-time APC also make it necessary that several SPIs be used. We base this assumption on the distribution of the computational load in the Lincoln Laboratory LPC vocoder design [7]. The real-time LPC implementation, although requiring very little memory for data buffering, requires three SPIs. The LPC and APC algorithms are of comparable complexity and three NEC μPD7720 SPIs, or possibly more, will most likely be necessary for APC.

When discussing the drawbacks of the Bell Labs DSP, we pointed out that most of the processing time required for computing the APC critical loops is spent accessing external memory. AMDF pitch estimation was the particular example cited. The Bell Labs DSP was ruled out because its I/O structure was not conducive to extensive use of external memory. A similar situation exists with the NEC μPD7720 SPI; however, it is less severe. The NEC μPD7720 possesses an equal amount of internal RAM as the Bell Labs DSP, and factors alluded to previously dictate that speech and other data be stored in external memory. An additional factor which makes the use of the NEC μPD7720 internal RAM undesirable is the internal memory pointer system that the NEC μPD7720 provides. It was discovered when programming the NEC μPD7720 for LPC [9] that a significant amount of overhead was devoted to

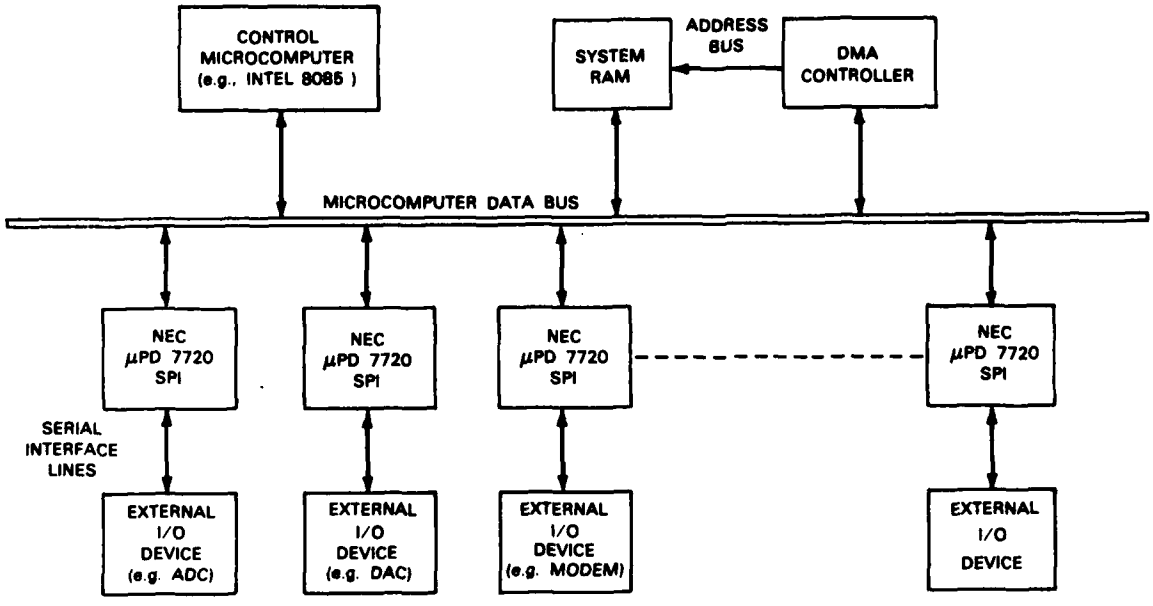


Fig. 2. A typical system architecture employing the NEC μPD7720.

conventional microprocessor/controller, including an 8/16-bit parallel I/O data port which attaches directly to a system data bus. The NEC μPD7720 allows this data port to be configured for DMA mode data transfers between it and other system devices. This DMA capability ideally permits speech and other data to be loaded into the NEC μPD7720 in 128 word blocks for parameter computation (see discussion below).

A typical system architecture employing several NEC μPD7720 SPIs is shown in Fig. 2. An architecture similar to this was implemented in the Lincoln Laboratory compact LPC vocoder, and we assert that an APC implementation based on the NEC μPD7720 SPI would also resemble the architecture shown in Fig. 2. A conventional microprocessor is employed as a system controller. In the figure, we have indicated that an Intel 8085 could serve in this function; however, a number of other commercially available microprocessors could be used as well. The primary purpose of the system controller is to manipulate the data paths among the multiple NEC μPD7720 SPIs in the system.

Although we have shown an indefinite number of SPIs being deployed in the system shown in Fig. 2, we can assume that at least two (most likely three) SPIs will be needed to implement APC. As was true of the Bell Labs DSP, the SPI possesses only 128 words of internal RAM which is insufficient for the data buffering requirements of APC. However, since the data throughput of the NEC μPD7720 SPI is considerably faster, it is possible to spread the memory requirements among the several SPIs in the system and have these devices pass data among themselves under the direction of a system wide controller. Another alternative is to deploy a separate

be read from external memory. Assuming that a frame consists of 160 samples and that 3 samples are skipped between summations, computing the AMDF for a single value of T would require 2x40x6.4 sec or 0.51 msec. Computing the AMDF for 60 values of T would require that 30.7 msec be spent in memory I/O alone. This is in excess of common speech analysis frame durations and, thus, demonstrates that using the Bell Labs DSP to implement APC in this fashion is infeasible.

3.2 The Nippon Electric Co. μPD7720 Signal Processing Interface

The NEC μPD7720 Signal Processing Interface (SPI) has been used previously at Lincoln Laboratory in a compact linear predictive vocoder implementation [7]. The success experienced with the NEC μPD7720 in this project prompted us to consider the NEC μPD7720 as a candidate for implementing real-time APC. The NEC μPD7720 architecture is described in [13] and features:

- -a 512 x 23-bit program ROM,
- -a 510 x 13-bit data coefficient ROM,
- -a 128 x 16-bit data RAM,
- -a 250 nsec 16 x 16-bit parallel multiplier which gives a 31-bit result.

Although the NEC μPD7720 has several characteristics in common with the Bell Labs DSP, significant advantages are apparent when the two DSPs are compared. These advantages include an enhanced I/O structure, interrupt service capabilities, and a 4-level stack which provides for up to 4-level nesting of subroutines. The I/O structure contains several features which enable the NEC μPD7720 to be easily interfaced with a

APC implementation is the necessity for transferring speech and other data between external memory and the limited on-chip RAM. The use of external RAM is essential because the 128 word internal RAM that the DSP provides is inadequate for storing the large speech buffers required for background computation of the APC side parameters. The DSP allows external memory to be substituted for the internal 1K program/coefficient ROM through a reconfiguration of the device and by using the multiplexed address/data bus which is brought off-chip. Unfortunately, external RAM cannot be substituted for the on-chip ROM because the DSP has no external memory write capability that directly utilizes this data bus.

The architecture of the DSP supports serial I/O with external devices through the use of asynchronous serial interface lines which could be used for transferring data to an off-chip memory. However, there are two problems associated with an approach which would utilize the serial data ports for memory I/O. The first problem is that the serial lines must be multiplexed between the memory and normal I/O devices, such as codecs and modems that would ordinarily communicate with the DSP. This problem could possibly be fixed by using external hardware that would arbitrate among these sources of data. The second, more critical problem, is data throughput. The following sample calculation determines the amount of time that would be required for serial I/O in computing an AMDF pitch estimate and illustrates the nature of the data throughput problem. At the maximum input clock rate, the DSP requires 400 nsec/bit to bring data on-chip. Therefore, reading a 16-bit word from memory would require 6.4 μsec. For each point in the AMDF summation, two speech samples, s[n] and s[n-T], must

3.1 AT&T Bell Laboratories DSP I

The digital signal processing integrated circuit initially considered in our study was the Bell Laboratories DSP I. The DSP has been successfully employed in other moderate complexity mid-rate coders at Bell Laboratories, such as Sub-band Coding [6] and ADPCM [4]. It therefore became a candidate for implementing real-time APC.

A complete description of the Bell Labs DSP architecture can be found in [5]. It features:

- a 1024 x 16 bit on-chip ROM for program and coefficient storage,
- a 128 x 20 bit on-chip data RAM,
- an extensive set of memory address registers and a separate address arithmetic unit,
- an Arithmetic and Logic Unit which features a 16 x 20 bit multiplier and a 40-bit accumulator.

The Bell Labs DSP architecture also features a great deal of parallelism which permits its relatively slow 800 nsec machine cycle time to be effectively reduced to 200 nsec by a 4-stage instruction pipelining mechanism. However, instruction pipelining is not always possible in many signal processing operations and is generally effective only in the type of computations required of digital filters and correlators (i.e, register transfers, multiplies and adds). Therefore, the Bell Laboratories DSP's 800 nsec cycle time is prohibitively slow for the implementation of many signal processing algorithms such as autocorrelation coefficient calculations and the other computations required in APC.

Aside from its relatively slow machine cycle time, we perceive that the major problem involved in using the Bell Laboratories DSP for real-time

properties of the auditory system [3]. Other efforts have included the development of various segmental quantization techniques that require that the quantization gain term, q, be computed several times per frame (on the order of 10) instead of just once as we have described [16]. The effect here, also, is to better control the properties of the quantization noise. Although including these techniques would strongly affect the execution of APC in any of the processors evaluated in this study, we felt that including them in the evaluation process would not provide further insight in assessing the relative performances of the DSPs.

3. DSP SELECTION

Before the existence of digital signal processor integrated circuits, DSPs employed in real-time signal processing architectures served primarily as number-crunching peripherals. In these systems, conventional microprocessors (e.g., the Intel 8085 and the Motorola 68000) served as central processing units that controlled the flow of data throughout the system and in and out of these peripherals. Although these implementations have been relatively small in size, smaller configurations have become possible by integrating more system control functions inside the DSP itself.

For the remainder of this section we will review the architectures of the AT&T Bell Laboratories DSP I, the Nippon Electric μPD7720, and the Texas Instruments TMS32010. We shall focus on each DSP's on-chip memory size and on the feasibility of supplementing this internal memory with external RAM. Secondly, we shall focus on each DSP's system control capabilities and evaluate its ability to manipulate the various data paths in a system.

2.1 Implementation Issues and Discussion

In a real-time implementation of APC, the side parameters T, α, a_i, and q are usually computed as background tasks while incoming speech is pre-emphasized and buffered by foreground I/O handling routines. This method of arranging the computations into background and foreground routines immediately imposes several requirements on the DSP that is used to implement the algorithm. The most obvious requirement is speed. In order to process speech data in real time, the DSP must be capable of executing a large number of machine instructions within the duration of a speech analysis frame. Another important issue is the size of the memory that the DSP either possesses on chip or can access externally. The memory requirements are large because computing much of the APC algorithm in the background demands that the speech data be double buffered. Multi-tasking foreground and background routines also requires a DSP capable of controlling a relatively large number of external I/O sources through the use of an interrupt mechanism. In the following section we shall see how these computational and control requirements of real-time APC translate into DSP architectural requirements.

In this section we have summarized some basic aspects of the APC algorithm. For the purpose of brevity, we have chosen not to describe several of the measures taken which improve APC's performance. For example, much of the research in Adaptive Predictive Coding has been involved with improving the performance of the predictive quantizer loop. These efforts include modifying the spectral predictor filter so that it shapes the quantization noise in ways which better match the masking

quality of the resynthesized speech in APC. In most APC implementations, pitch period estimates are obtained using either autocorrelation analysis or the average magnitude difference function (AMDF) [14]. In the autocorrelation method, the autocorrelation function is computed for each frame of speech. The distance in lags between its peaks is taken as the pitch estimate. In the AMDF method, the method most often employed in real-time APC, the average magnitude difference function is substituted for the autocorrelation function and is computed in a manner very similar to the autocorrelation function for each frame of speech. The distance between nulls in the AMDF is taken as the pitch estimate. In this study we have used the AMDF method and have employed the following technique. To limit the number of computations, only certain values of T, T_i, corresponding to fundamental frequency values in the range from 50 to 400 Hz are used. These values of T_i are precomputed and stored in a table. The AMDF, $D[T_i]$, rewritten as

$$D[T_i] = \sum_{n=0}^{N-1} \left| s[n] - s[n-T_i] \right| \tag{5}$$

is computed for each value of T_i. The value of T_i which gives the minimum AMDF value is taken as the pitch estimate. Another measure often taken to minimize the number of computations is to compute the AMDF summation for every fourth sample of s[n] and s[n-T], skipping three samples in between. We have also taken this step to minimize the computation time.

samples. The transmission frame size for this sampling rate is 192 bits which are allocated among quantized residual and side parameters as shown in Table I.

4.2 Critical Loop Timing and Control Strategy

The first step actually taken in the evaluation of the TMS32010 was a critical loop timing study in which the various portions of the APC algorithm were coded in the TMS32010 instruction set and approximate execution times of the code were calculated. This critical loop timing study was useful in obtaining two types of information. First, it has given us some benchmark timing figures which could be used as an objective measure for comparing the TMS32010 against the other digital signal processing integrated circuits. Secondly, this timing data provided information which was later used in decisions affecting the hardware design.

Two versions of several of the critical loops were coded. In version I of the code, **TBLR** and **TBLW** instructions were used to transfer data to and from off-chip memory. In version II, the data port I/O instructions, **IN** and **OUT**, were used to transfer data to external RAM. We have summarized the execution times of the software units in Table II. All of these execution times are based on a 200 nsec machine cycle time. Listings of the code written for these critical loops appear in the appendices. From examining the execution times of the critical loops in Table II, it is apparent that the code which incorporates the **TBLR** and **TBLW** mode of external memory access could not execute within a 20 msec frame duration.

TABLE I

BIT ALLOCATION PER FRAME

Quantity		Bits/ Frame
d[n]	Residual	157
T	Pitch	6
α	Pitch Predictor Coefficient	4
q	Quantizer Level	5
k_1	Reflection Coefficients	5
k_2		5
k_3		5
k_4		5
TOTAL		192

TABLE II

SUMMARY OF CRITICAL LOOP EXECUTION TIMES

OPERATION	VERSION I		VERSION II	
	Execution* Time (msec)	% Real Time	Execution* Time (msec)	% Real Time
AMDF PITCH ESTIMATION	10.80-11.00	54.0-55.0	6.60	33.0
ALPHA CALCULATION	.92	4.6	.62	3.1
1st RESIDUAL CALCULATION & LPC AUTOCORRELATION ANALYSIS	3.18	15.9	2.92	14.6
REFLECTION COEFFICIENT CALCULATIONS	.16	.8	.16	.8
PREDICTIVE QUANTIZER	2.84	14.2	1.40-2.16	7.0-10.8
RECEIVER LOOP	2.00	10.0	1.60	8.0
ADC-DAC I/O	1.09-1.28	5.4-6.4	1.09-1.28	5.4-6.4
TRANSMIT MODEM I/O HANDLER**	1.40	7.0	1.4	7.0
RECEIVE MODEM I/O HANDLER**	1.40	7.0	1.4	7.0
TOTAL	23.79-24.18	119.0-120.9	17.19-18.14	86.0-90.7

*Execution time per frame
**These are foreground routines. Execution times were calculated by multiplying the per sample execution times by the 160 sample frame size.

Thus, if only one TMS32010 were used, we would not expect the algorithm to execute in real time. One therefore would have to partition the APC algorithm among two or more TMS32010 DSPs. On the other hand, if the I/O ports are used for transferring data to external memory in conjunction with external memory address generators, it seems possible that a single TMS32010 would be all that is needed.

During our study we briefly examined the trade-offs between implementing a single TMS32010-based APC processor versus one which incorporates two TMS32010 DSPs with version I of the APC software partitioned. Although a dual TMS32010-based processor possesses potentially more processing power, it is difficult to make effective use of it due to interprocessor communication overhead. In designing a dual TMS32010 architecture, the first task is to find a reasonable partition of the APC software between the two TMS32010 DSPs. The most straightforward partition, a direct split between the analyzer and synthesizer, would result in an unbalanced distribution of the computational load. The analyzer requires a significantly greater proportion of the computational resources. In fact, given the execution times of the analyzer loops, it is improbable that a single TMS32010 would be able to execute all of the analyzer routines within the 20 msec frame duration. Therefore, a more uniform partitioning of the APC algorithm, in terms of computational requirements, is needed. This alternative has a more subtle drawback in terms of data communication overhead. Although the APC analyzer software can be segmented into several autonomous units, practically all of these units process the same speech data. If these units are contained in

separate TMS32010s, then either entire speech buffers would have to be passed among DSPs or each of the DSPs would have to access identical copies of the same speech data. The first option entails a significant amount of processing time being devoted to I/O among the processors. The second option would require that either memory be shared or data be copied to both TMS32010 processors. Both of these memory management schemes are unduly complex.

After recognizing the difficulties involved in using a dual TMS32010 system, we decided not to pursue this effort and, instead, adopted the single-chip design which uses the I/O ports for transferring data to external memory. For the remainder of this section and the next, we describe a software control strategy and external memory allocation for this one chip design. We use the term, software control strategy, to refer to the method used to combine software units. Our philosophy in adopting a software control strategy in this APC implementation has been to relegate as much of the computation to background tasks as possible. This allows the foreground routines, which are executed upon interrupt from the external I/O sources, to be simple I/O handlers that merely control the pointers required for buffering the data. In Table II, foreground routines are identified with two asterisks.

The obvious disadvantage of computing the APC routines as background routines is the overall increased demand for memory. However, as we shall illustrate in the following section, the memory requirements of this APC implementation fit safely within the confines of commercially available RAMs.

4.3 Memory Allocation

In Fig. 3 we show how memory is allocated among the APC software units. A total of 2048 words of RAM are required and have been divided among eight 256-word pages. The 256-word page size is used to accommodate the use of 8-bit external address generators which are used as memory pointers as described in the next section. Computing the analyzer routines as background tasks normally requires that the incoming speech be double buffered. Actually, triple buffering is used. The extra buffer, stored on a separate page, is provided for storing the previous pitch period of speech that is necessary in computing the pitch period estimate, T, the pitch predictor coefficient, α, and the first residual signal, $e_1[n]$. During these calculations, the speech sample s[n-T] is needed and, depending on the value of T, could reside on the previous pitch period page. The two buffers of input speech used in these background computations, the current processing frame and the previous pitch period, are arranged contiguously on pages 0 and 1 so that the same 8-bit memory pointer, with the addition of a 9th bit used for page crossing, can be used to access these two pages of data as a single 512-word block. We have thus eliminated the overhead in software involved in page crossing. The 9th bit of the pointer to speech sample s[n-T] is set by the hardware when it reaches the end of page 0. A separate pointer to the speech sample s[n] is initialized to the bottom of page 1 and never crosses the 0/1 or the 1/2 page boundaries.

The reconstructed speech signals in the analyzer and synthesizer (i.e., the state space of the recursive pitch prediction filters) are

MEMORY PAGE	FUNCTION
0	PREVIOUS PITCH PERIOD OF SPEECH DATA
1	CURRENT SPEECH PROCESSING FRAME
2	RECONSTRUCTED SPEECH FOR ANALYZER PREDICTIVE QUANTIZER LOOP
3	RECONSTRUCTED SPEECH FOR SYNTHESIZER LOOP
4	SPEECH INPUT BUFFER FROM ADC
5	SYNTHESIZER OUTPUT BUFFER NO. 1
6	SYNTHESIZER OUTPUT BUFFER NO.2
7	DOUBLE BUFFERED RESIDUAL OUTPUT (Analyzer)
	DOUBLE BUFFERED RESIDUAL INPUT (Synthesizer)
	DOUBLE BUFFERED APC SIDE PARAMETERS (Analyzer)
	DOUBLE BUFFERED APC SIDE PARAMETERS (Synthesizer)

136 392-N

Fig. 3. RAM allocation for APC implementation.

stored in pages 2 and 3 which are implemented in hardware as circular buffers. The 256-word buffer size is more than adequate for storing the state space which has a maximum length of 160 points.

Since we have decided to compute the predictive quantizer and receiver loops as background routines, the single bit/sample residual data must also be double buffered, as do the APC side parameters. However, since this residual data stream is serial, we can reduce its storage requirements by packing it into 16-bit words. This data is stored on page 7, along with the APC side parameters.

We have eliminated the need in the analyzer for buffering the first residual signal by combining the autocorrelation computation with the calculation of the first residual signal. Through the use of a first-in first-out buffer maintained in internal RAM (see code in the appendix) for storing only P+1 first residual values, we are able to compute the first residual autocorrelation values that are necessary for computing the LPC spectral predictor coefficients directly from the speech signal.

ROM is needed for storing program instructions and data constants. Although we have not completely specified the amount of ROM which will be needed, we have assumed that no more than 2048 words of ROM will be required. The hardware implementation of both ROM and RAM will be discussed in the following section.

4.4 Hardware Design

There are two principal features of the APC processor hardware. The first is a high speed external memory interface circuit. This circuitry provides two separate memory address generators which are operated under

programmed control of the TMS32010. The second major feature is an interface between the TMS32010 and four external I/O devices (the analog to digital and digital to analog converters, the transmit modem, and the receive modem). This interface allows the external devices to communicate with the TMS32010 CPU on an interrupt basis. For the remainder of this section, we outline our approach for designing the APC processor hardware.

A functional block diagram of an APC processor architecture is shown in Fig. 4. In this figure, we have labeled the external memory controller circuit and the external I/O interface portions of the architecture explicitly. The architecture permits access to external memory from the TMS32010 under the two modes described in the previous sections, using either the memory address bus in conjunction with the **TBLR/W** instructions or the port address bus for faster access. The memory address bus is used primarily for fetching program instructions and constants from ROM via mode I. Under mode II, the data stored in locations 0-1023 of RAM are designed to be accessible using the address generation logic which is contained in the external memory controller circuitry. In order to retain maximum flexibility, we have made all 2K of RAM accessible to the TMS32010 under both modes by multiplexing the address bits input to the RAM devices.

The data ports can be thought of as eight physical ports which are directly tied to the TMS32010. In actuality, data transfers involving the ports will utilize the data bus as well. A 3-bit port address (PA0-3) is decoded and is used to select one of eight devices which is to send or receive data from the TMS32010 over the bus. In the proposed system shown in Fig. 4, three of the eight ports are used to interface the TMS32010 to

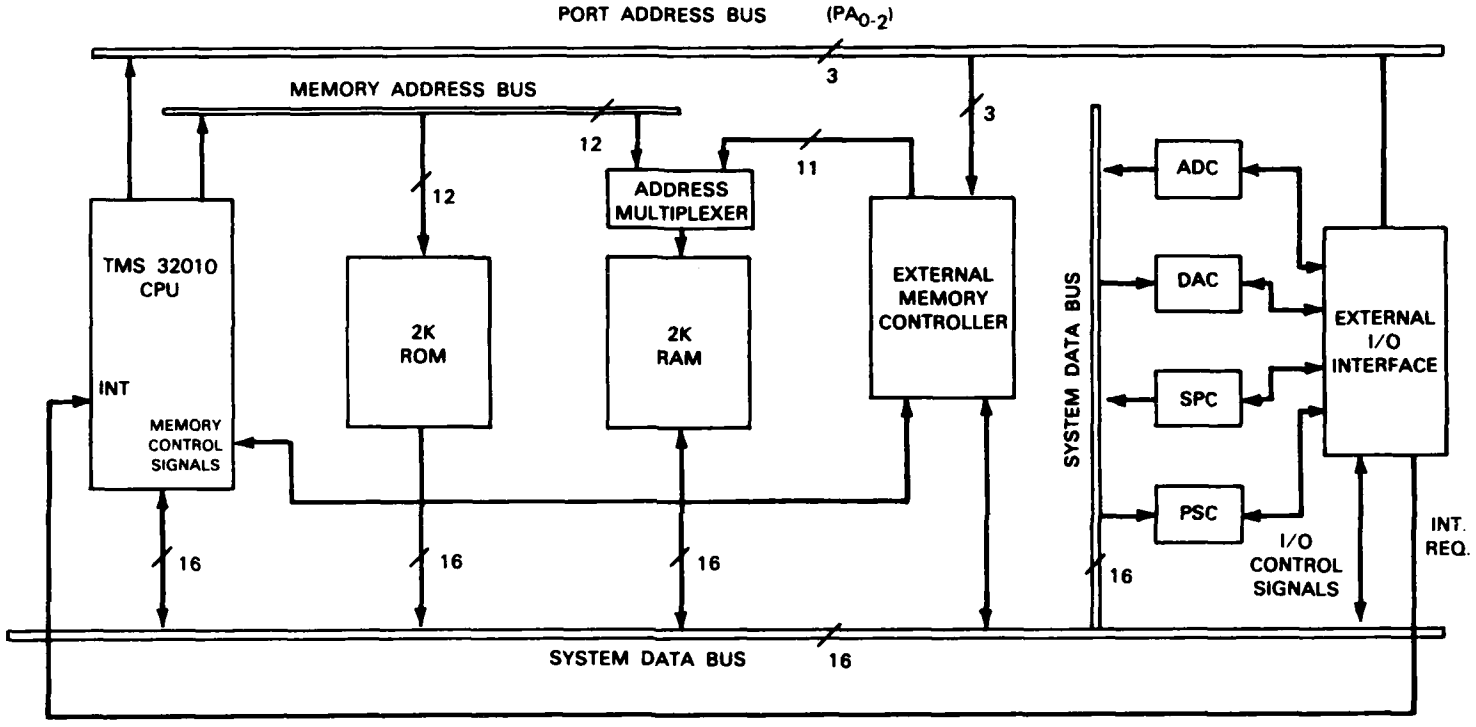


Fig. 4. Architecture of the proposed APC processor.

136391-N

external I/O devices. The I/O devices requiring separate ports are the analog to digital converter (ADC), the digital to analog converter (DAC), and the parallel to serial converter (PSC) which subsequently connects to a serial transmit modem. We have been able to input data from the serial receive modem without the use of an explicit I/O port. An additional port is used to interface the TMS32010 with external interrupt control logic. The other four ports are used to interface the TMS32010 with external RAM.

4.4.1 Hardware Design-External Memory Controller

Figure 5 is a more detailed schematic of the External Memory Controller circuit. In this schematic we have shown explicitly the memory I/O control signals which are generated by the TMS32010 CPU, the logic used for decoding these signals, and the port/memory address bus. For flexibility in the software design, external memory can be used as either a 4K word block, consisting of both ROM and RAM to be accessed under mode I type memory transfers (i.e., by executing **TBLR** and **TBLW** instructions), or as separate ROM and RAM each consisting of 2K words. In mode II, the first 1K words of RAM are accessed via the data I/O ports in conjunction with the external address generation logic which shall be described below. These two modes of memory access are distinguished in the hardware by decoding the TMS32010 control signals **MEN**, **DEN**, and **WE**, along with the memory address bit A_{11}.

For the most part, mode I memory transfers are used primarily for fetching program instructions and data constants from ROM. However, we also must access pages 4 through 7 of RAM under mode I. For mode I, the 4K [illegible] of memory has been partitioned into a lower 2K section of ROM and an

upper 2K section of RAM. A simple 2-level hardware decoding of address bit A_{11} distinguishes a read from ROM from a read from RAM. For a read operation from ROM, the TMS32010 signal **MEN** will become active low, along with address bit A_{11}. These two signals cause the **ROM-ENABLE** signal to become active low, which is directly tied to the chip-select (CS) inputs of the ROM devices. If a read from RAM is to take place, **MEN** will again be active low, but A_{11} will be high since RAM is contained in the upper 2K section of the memory address space. The address bit A_{11} is inverted and combined with **MEN** to generate chip select signals for the RAM devices (see Fig. 5).

Modes I and II memory transfers are distinguished by decoding the address bit, A_{11}, along with the TMS32010 control signals **WE** and **DEN**. These signals are both active low and are combined with A_{11} to generate a **PORT-ENABLE** signal (also active low) which enables a 3-to-8 line decoding of the port/address bus which is assumed to contain a valid 3-bit port address. The decoder circuit signals to one of the eight devices tied to its output lines to communicate with the TMS32010 CPU over the data bus. Two separate DMA controllers are being employed as external memory address generators. In the schematic in Fig. 5, Advanced Micro Devices (AMD) Am2940s [1] are used. These particular devices have been chosen primarily because of their speed. The machine cycle time of the TMS32010 is nominally 200 nsec, and a reasonable, but fast, access time for commercially available RAMs is 50 nsec. According to the specifications given for the Am2940, its propagation delay, combined with the delays of the other combinational logic in the external memory interface, provide

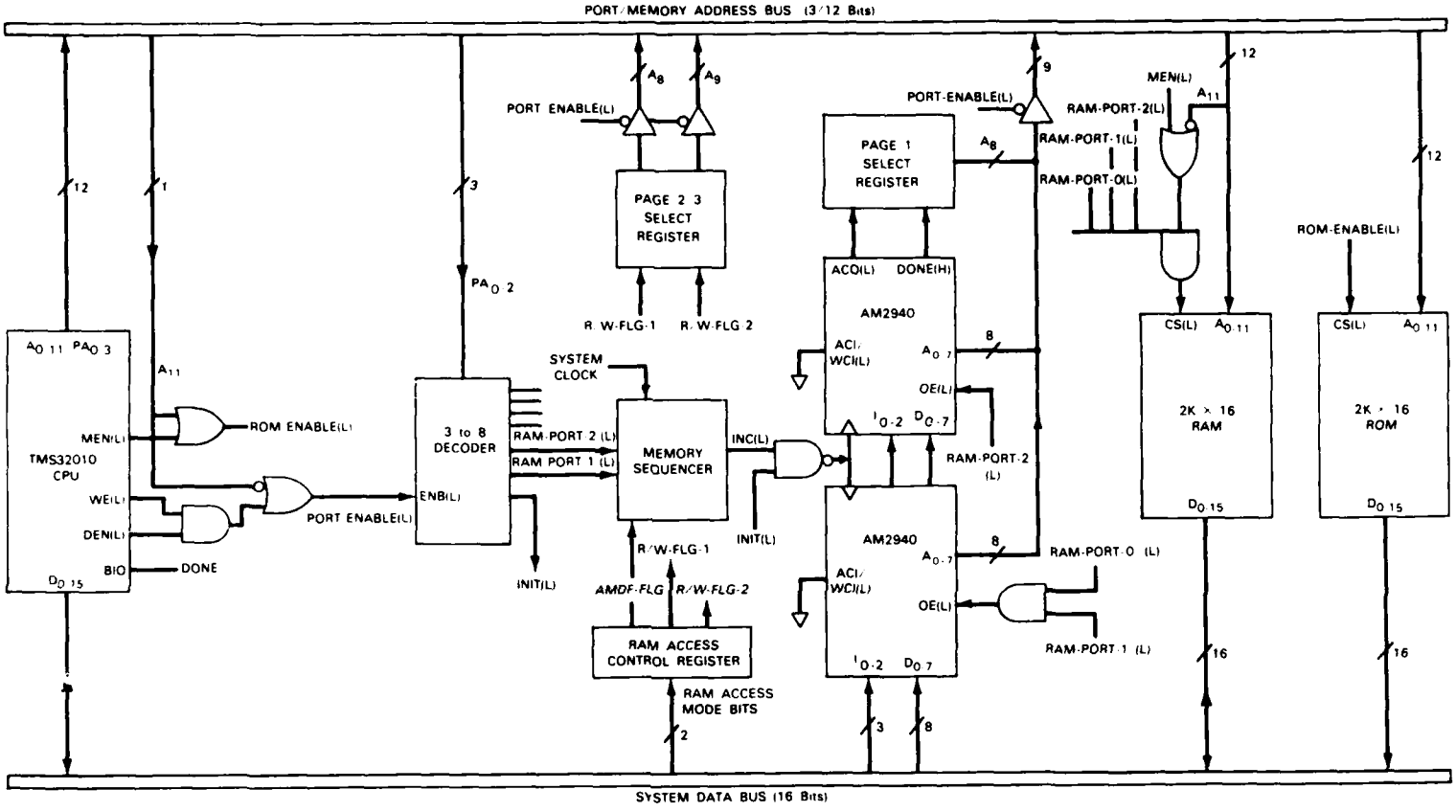


Fig. 5. Schematic of the external memory interface circuit.

idequate time for data being accessed from RAM to settle on the data bus before the end of the TMS32010 memory read cycle. Similar time constraints are met for the memory write cycle.

The Am2940s are programmable and receive instructions from the TMS32010 over the data bus via the port I/O mechanism. One of the eight data ports is dedicated entirely to providing initialization and other instructions to the Am2940s. When this port is selected, the **INIT** signal becomes active low (see Fig. 5) and the Am2940s receive instructions over the data bus. The format of the data instructions which are given to the Am2940s is described below.

Although the memory requirements of the APC algorithm are extensive, an advantage that the algorithm provides is that memory access is primarily sequential within a page. In other words, speech samples and other data that are used within the same software routine will generally reside on the same page and will be arranged sequentially within that page. This way, after the DMA controllers have been programmed at the beginning of a routine, there is little interaction between the TMS32010 CPU and the address generators during the remainder of the routine's execution. In addition, most of the computationally intensive signal processing routines involve sequential data fetches from two memory locations. The Am2940 address generators will increment their present addresses after an **INC** signal is generated by the Memory Sequencer circuit shown in Fig. 5. The Memory Sequencer is a relatively simple Finite State Machine (FSM) which ensures that the memory pointers to RAM are incremented by the proper amount after each data transfer. Accessing the I/O ports using

```
        ZALS    AMDF-L, Ø           If current AMDF is smaller
        ADDH    AMDF-H,Ø            make it the new minimum
        SACL    MIN-AMDF-L,Ø
        SACH    MIN-AMDF-H,Ø
        ZALS    *,ARØ, Ø
        SACL    T,Ø                 Save the present pitch value

SAME    MAR     *+,ARØ              Loop to next pitch value
        ZALS    NUM-PITCHS,Ø
        SUB     ONE,Ø
        SACL    NOM-PITCHS,Ø
        BGEZ    LOOP-1
```

APPENDIX I

AMDF PITCH ESTIMATION - VERSION I

```
*
*  Computes a pitch estimate from one out of 60 values which are stored in a
*  table in internal RAM.  Assumes speech data to be stored in external RAM
*  accessed using TBLR

INIT    ZALS    BIG-NUM-L,Ø
        ADDH    BIG-NUM-H,Ø
        SACL    MIN-AMDF-L,Ø        MIN-AMDF maintains min AMDF
        SACH    MIN-AMDF-H,Ø           value, init it to something large
        LACK    #60
        SACL    NUM-PITCHS,Ø        Num-Pitchs is loop counter

        LARK    ARØ,#P-TBL-ADDR     ARØ points to pitch table

LOOP-1  LACK    #S-ADDRR            Initialize pointers to speech
        SACL    S-PTR, Ø               Data s[n] and s[n-T]

        SUB     *,ARØ,Ø             Compute pointer to s[n-T] by
        SACL    ST-PTR,Ø               subtracting away current pitch
        ZAC
        SACL    AMDF-L,Ø            Initialize Accumulated AMDF
        SACH    AMDF-H,Ø               value
        LARK    ARØ, #NUM-SAMPLES   ARO is loop counter


LOOP-2  ZALS    S-PTR, Ø            Load ACC w/ pointer to s[n]
        TBLR    S                   Read s[n]
        ADD     THREE, Ø            Update pointer skipping
        SACL    S-PTR, Ø            Three speech samples
        ZALS    ST-PTR, Ø           Do same w/ s[n-T]
        TBLR    ST
        ADD     THREE, Ø
        SACL    ST-PTR, Ø

        ZALS    S, Ø              Compute /s[n] - s[n-T]/
        SUB     ST,Ø
        ABS
        ADD     AMDF-L,Ø            Update AMDF value
        ADD     AMDF-H,15
        SACL    AMDF-L,Ø
        SACH    AMDF-H,Ø
        MAR     *-,AR1              If not finished w/ current
        BNAZ    LOOP-2                pitch value, loop

        ZALS    MIN-AMDF-L,Ø        Compare current AMDF value with
        ADDH    MIN-AMDF-H,Ø        previous minimum
        SUB     AMDF-L,Ø
        SUB     AMDF-H, 15          If current is larger, loop
        BLZ     SAME                to next pitch value
```

[15] Texas Instruments, TMS320 Data Manual.

[16] R. Viswanathan, W. Russell, and A.W.F. Huggins, "Design and Real-Time Implementation of a Robust APC Coder for Speech Transmission over 16Kbps Noisy Channels," Bolt Beranek and Newman Inc., BBN Rep. 4565 (December 1980).

References

[1] Advance Micro Devices, "Bipolar Microprocessor Logic and Interface," 1983 Data Book.

[2] B.S. Atal and M.R. Schroeder, "Adaptive Predictive Coding of Speech Signals," Bell System Technical Journal 55:1973-1985 (1975).

[3] B.S. Atal and M.R. Schroeder, "Predictive Coding of Speech Signals and Subjective Error Criteria," in Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, IEEE, Tulsa (1978).

[4] J.R. Boddie, J.D. Johnson, C.A. McGonegal, J.W. Upton, P.A. Berkley, R.E. Crochiere and J.L. Flanagan, "Adaptive Differential Pulse-Code-Modulation Coding," Bell System Technical Journal 60: No. 7 1547-1561 (September 1981).

[5] J.R. Boddie, G.T. Daryanani, I.I. Eldumiati, R.N. Gadenz, J.S. Thompson, and S.M. Walters, "Digital Signal Processor: Architecture and Performance," Bell System Technical Journal 60: No. 7, 1449-1462 (September 1981).

[6] R.E. Crochiere, "Sub-Band Coding," Bell System Technical Journal No. 60: 1633-1653 (September 1981).

[7] J.A. Feldman, "LPC Chip Set User's Guide," Project Report, PSST-1, Lincoln Laboratory, M.I.T. (November 1982).

[8] J.A. Feldman, Personal Communication.

[9] E.M. Hofstetter, Personal Communication.

[10] J. LeRoux and C. Gueguen, "A Fixed Point Computation of Partial Correlation Coefficients in Linear Prediction," in Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, IEEE (1977).

[11] J. Makhoul, "Linear Prediction: A Tutorial Review," Proceedings of the IEEE 63 (12) (April 1975).

[12] J.D. Markel and A.H. Gray, "Linear Prediction of Speech" (Springer-Verlag 1976).

[13] Nippon Electric Company, NEC 7720 Data Manual.

[14] M.J. Ross, H.L. Shaffer, A Cohen, R. Freudberg and H.J. Manley, "Average Magnitude Difference Function Pitch Extractor," IEEE Transactions on Acoustics, Speech, and Signal Processing ASSP-22(5) (October 1974).

Acknowledgment

The author wishes to acknowledge the assistance of E. Hofstetter, E. Singer, and J. Feldman throughout this investigation. They have provided helpful and insightful comments and criticism and, in the case of Hofstetter and Singer, have carefully edited earlier drafts of this report.

which allowed us to obtain some benchmark timing figures which were used to characterize the TMS32010. These timing figures also helped us in making some hardware decisions and allowed us to determine the approximate hardware requirements.

There are several steps which can follow from this work. The most logical step would be for the architecture outlined in Section 4 of this report to be constructed. Although a preliminary hardware design was given in this report, many of the details still need to be more fully developed.

acknowledgment procedure as follows. The bit is reset by the TMS32010 after the contents of the ICR are read over one of the I/O ports and a word is written back with the corresponding bit set to 1.

5. SUMMARY

In this report we have given the results of an APC processor design study. The system that we have proposed is based on the Texas Instruments TMS32010. We began by outlining the basic features of the algorithm and pointing out those aspects of the algorithm which make its implementation challenging. The major problem associated with implementing APC in real time is memory. We acknowledged the necessity of computing the APC side parameters as background tasks which inherently requires an extensive amount of RAM.

As a part of this study, we examined two other DSPs besides the TMS32010: the AT&T Bell Laboratories DSP I and the Nippon Electric μPD7720 Signal Processing Interface. We have compared these DSPs against the TMS32010. The Texas Instruments TMS32010 was determined to be most suitable for a real time implementation of APC because of its speed, its ability to perform the numerically intensive signal processing operations required by APC, and its relatively sophisticated control features which enable it to handle the memory addressing and I/O requirements of APC.

The objective of this study has been two-fold. We wanted to learn the relative strengths and weaknesses of the three DSP ICs, and we wanted to determine whether a compact implementation of APC and other moderate bit rate speech coders of comparable complexity were feasible using the TMS32010. Towards these ends a critical loop timing study was performed

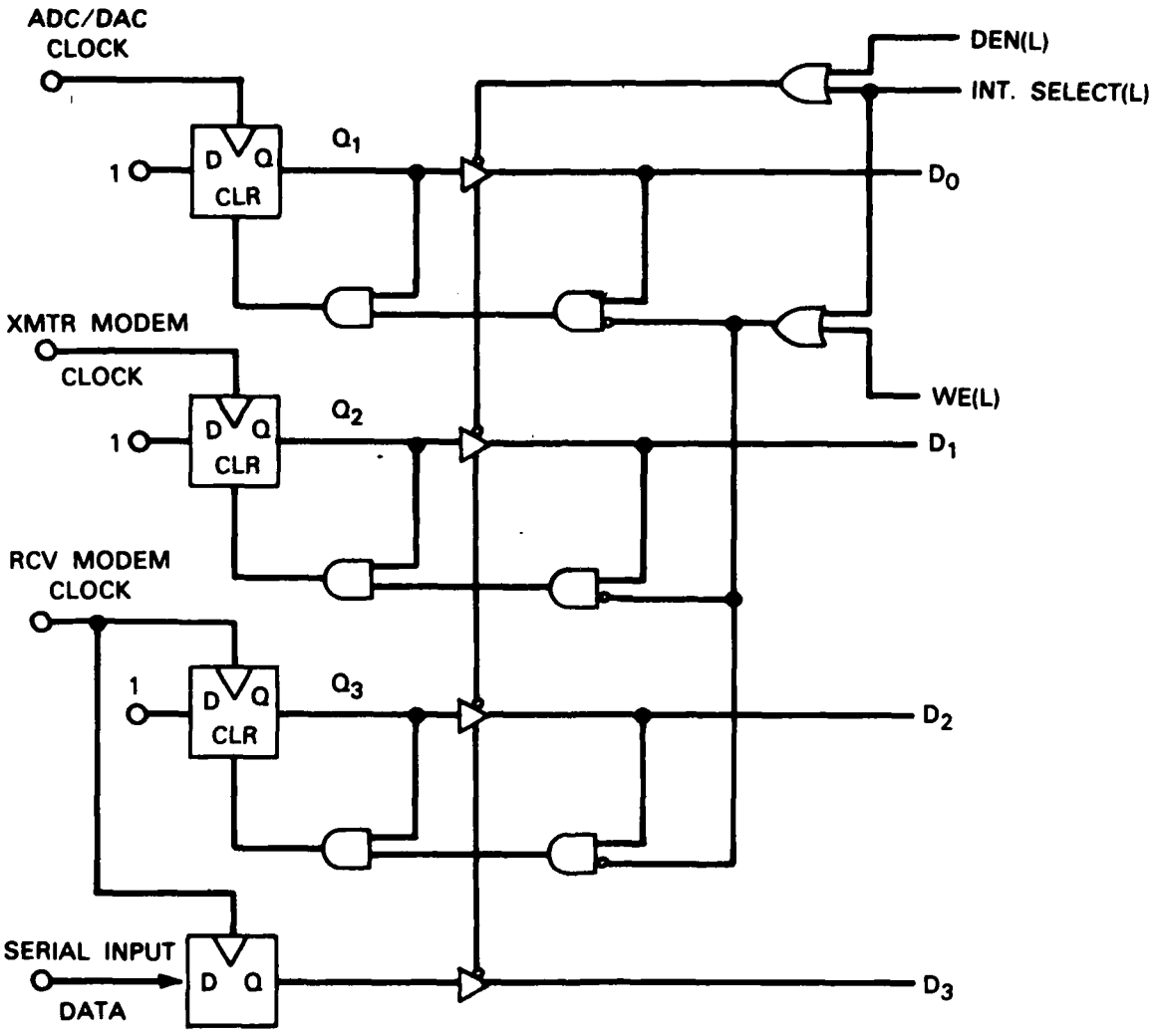


Fig. 9. Interrupt control register.

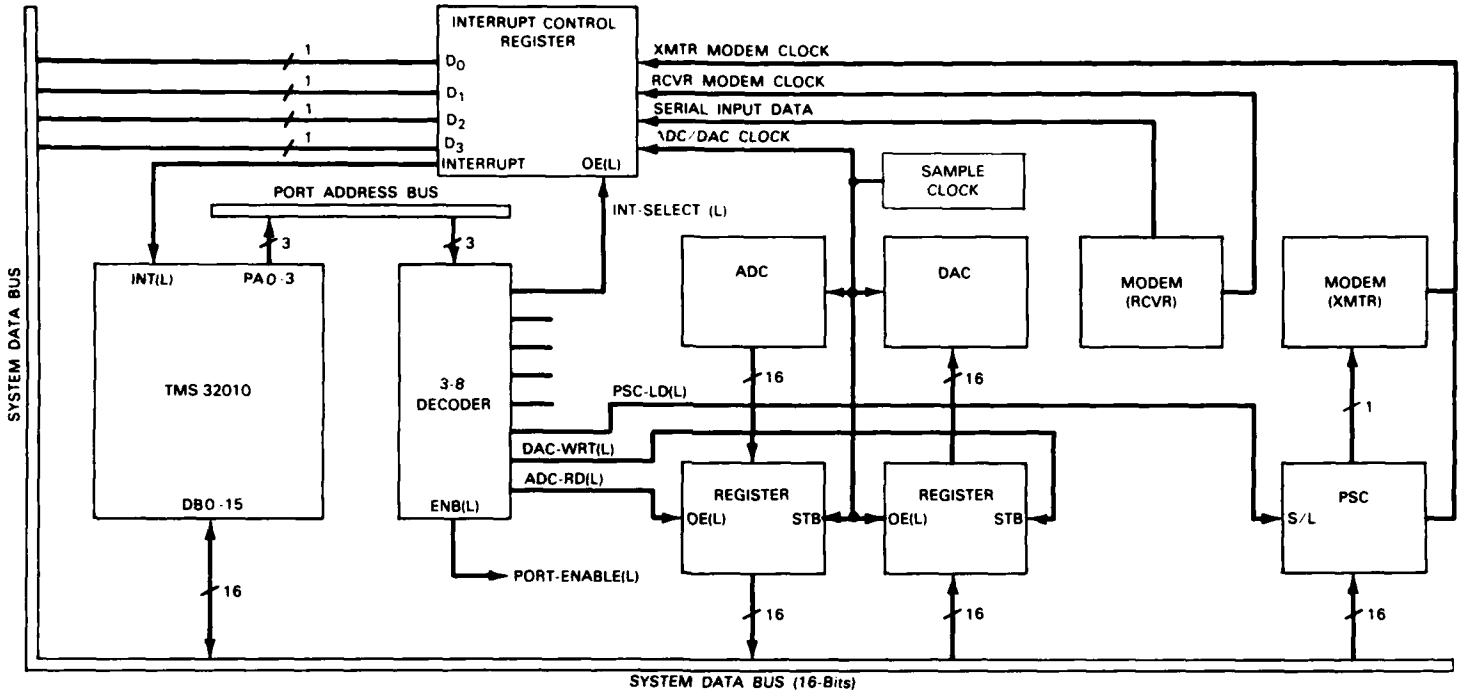


Fig. 8. External I/O device interface circuit.

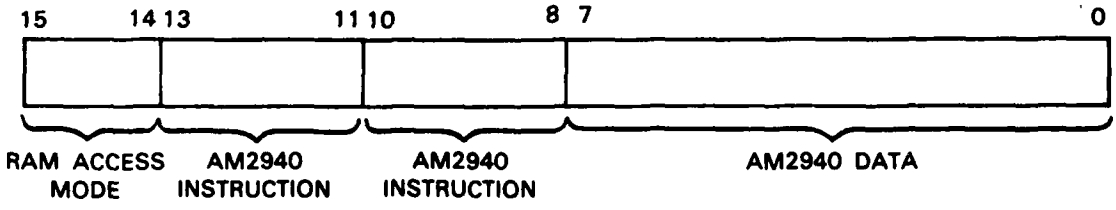
136394-N

control register which will uniquely indicate the presence of an interrupting device by one of its bits being set. The register is read by the TMS32010 any time after the interrupt has occurred.

The I/O interface circuit is shown in Fig. 8. Buffer registers are provided between the TMS32010 and the analog to digital converter (ADC) and the digital to analog converter (DAC). These registers allow the data transferred between the TMS32010 and these devices to be double buffered, eliminating the need for the TMS32010 to be directly involved in any type of handshaking procedure. A parallel to serial converter (PSC) is provided between the transmit modem and the TMS32010. The parallel to serial converter changes the parallel data output from TMS32010 into a serial bit stream appropriate for the transmit modem. It also serves as a data buffer as well.

The interrupt control register (ICR) is shown in Fig. 9. It multiplexes three externally generated I/O control signals on to the single interrupt line of the TMS32010 through the use of a single logic gate. These three control signals are the ADC/DAC sampling clock, the transmit modem clock, and the receive modem clock. The ICR is actually a 4-bit register, with the fourth bit being used to store the serial input data from receive modem. The ICR is implemented as a set of four D-latches. A typical interrupt service scenario would proceed as follows. When one of the external devices interrupts the TMS32010, the interrupt signal is passed on directly to the TMS32010 when the corresponding bit is set in the ICR. A bit set in the ICR prohibits the interrupting device from reinterrupting the TMS32010 until the bit is reset through an

(a)



(b)

ACCESS MODE		MEMORY PAGE AND FUNCTION
BIT 1	BIT 0	
0	0	PAGES 0 & 1 AMDF MODE ACCESS (R)
0	1	PAGES 0 & 1 NON AMDF MODE (R)
1	0	PAGE 2 (R/W)
1	1	PAGE 3 (R/W)

136387-N

Fig. 7 (a). External memory interface program instruction word.
(b) RAM access mode bits.

the FSM will return to the IDLE state. If an AMDF pitch estimate is being computed, the FSM will continue to its INCREMENT #2 and INCREMENT #3 states, incrementing the Am2940s three more times in the process.

The external memory interface circuit is programmed by issuing a sequence of 16-bit instruction words over one of the I/O ports. Each instruction word is broken into several fields which are labelled in Fig. 7(a). The least significant eight bits contain the initialization data for the Am2940s (e.g., initial addresses, etc.). Bits 8 through 10 and 11 through 13 contain the Am2940 instructions, and bits 14 and 15 contain the RAM access mode bits which are described in Fig. 7(b). A list of the Am2940 program instructions is given in [1]. The appropriate setting of the RAM access mode bits indicates to the external memory controller which memory page is to be accessed and the number of times the memory pointers are to be incremented during each memory I/O cycle. The access mode bits will control the setting of the **R/W-FLG-1**, **R/W-FLG-2** and the **AMDF-FLG** signals, which are output from the RAM Access Control Register and are input to another register which selects pages 2 and 3 of RAM, and are also input to the memory sequencer FSM. A table is provided in Fig. 7(b) that summarizes the settings of the RAM access mode bits.

4.4.2 Hardware Design-I/O Interface Circuit

The second major task in the hardware design is to interface the four external I/O devices with the TMS32010. These external devices are to communicate with the TMS32010 on an interrupt basis. Since the TMS32010 has only one interrupt line, the control signals output from these I/O devices must be multiplexed. Our approach has been to design an interrupt

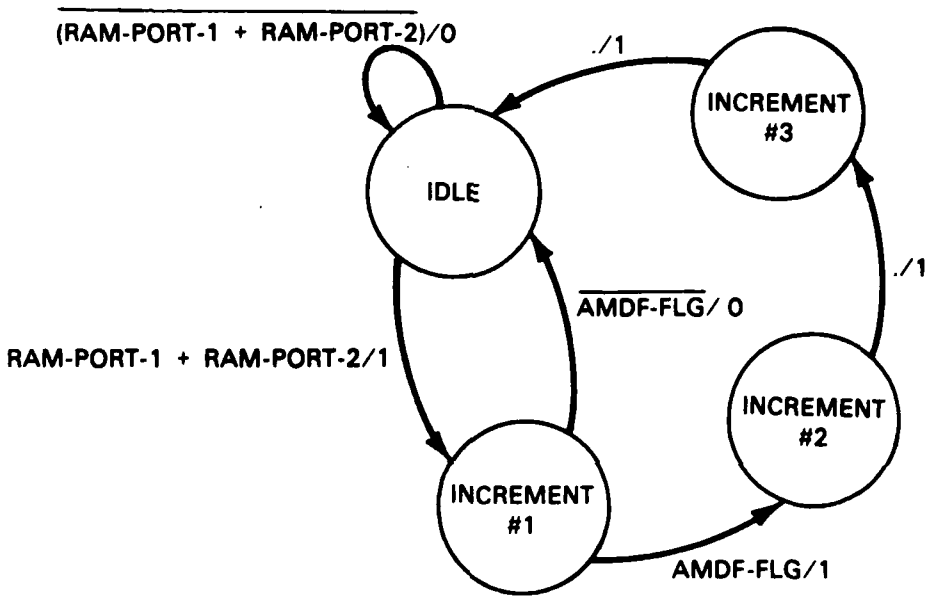


Fig. 6. Memory sequencer state transition diagram.

either **RAM-PORT-1** or **RAM-PORT-2** causes both addresses to increment while an access via **RAM-PORT-0** causes no incrementation. Thus, a typical autocorrelation computation would require first reading s[n] through **RAM-PORT-0** and then reading s[n-i] through **RAM-PORT-1** or **2**. Since address incrementation occurs following the second read, subsequent data fetches will access the proper data. All of the routines which access external memory using the I/O ports have the memory pointers incremented once after each transfer, the exception being the routine which computes the AMDF pitch estimate. In this routine, three speech samples are skipped between each point in the AMDF summation (see Eq. 5); and the memory pointers must, therefore, be incremented three times.

The state transition diagram for the memory sequencer FSM is shown in Fig. 6. Shown in the figure are three inputs to the FSM, **RAM-PORT-1**, **RAM-PORT-2** and **AMDF-FLG**, in addition to the system clock. The output of the FSM is an **INC** control signal which causes both of the Am2940 address generators to increment their memory pointers. The memory sequencer FSM steps through its sequence while the CPU is processing the data it has just read, or, in the case of a memory write, while the CPU is preparing to output another word to RAM. We have managed to save a considerable amount of time by having these operations carried out concurrently. The memory sequencer FSM normally sits in an idle state. After the CPU has finished its read or write cycle, signalled by either the **RAM-PORT-1** or the **RAM-PORT-2** signals becoming inactive, the FSM will enter the INCREMENT #1 state. During the transition it generates the **INC** control signal which is tied to both Am2940s. If the AMDF pitch estimation is not being performed,

APPENDIX II

AMDF PITCH ESTIMATION - VERSION II

```
*
*  Compute a pitch estimate from one of 60 values stored in a table in internal
*  RAM.  Assumes speech data to be stored in external RAM which is accessed via
*  the data ports using the IN instruction.  External memory interface is initial-
*  ized by a separate initialization routine.

INIT      ZALS      BIG-NUM-L,Ø          Initialize minimum AMDF value to some
          ADDH      BIG-NUM-N,Ø            big number
          SACL      MIN-AMDF-L,Ø
          SACH      MIN-AMDF-H,Ø
          LACK      #60                  Initialize counter for number
          SACL      NUM-PITCHS,Ø           of pitch values
          LARK      ARØ,#P-TBL-ADDR      Initialize ARØ to point to pitch table
          ZAC                            Clear RAM access control word
          SACL      ACCESS-CTR-WD,Ø        to indicate AMDF mode in
          CALL      EXT-MEM-INIT           initialization of external memory
                                           interface

LOOP-1    LACK      #ADDR-S              Init external pointers to s[n] and s[n-T]
          SUB       *,Ø,ARØ                Issue Reinitialization Instructions to
          ADD       REIN-INS,11            AM2940's
          ADD       LOAD-INS,8
          SACL      INSTR,Ø
          OUT       INSTR, INTRFC-PORT     Output Instructions over Data
                                             Port

LOOP-2    IN        S, RAM-PORT-1        Read s[n] and s[n-T] from external RAM
          IN        ST, RAM-PORT-2
          ZALS      S,Ø                  Compute /s[n]-s[n-T] /
          SUB       ST,Ø
          ABS
          ADD       AMDF-L,Ø             Update AMDF Value
          ADD       AMDF-H,15
          SACL      AMDF-L,Ø
          SACH      AMDF-H,Ø
          BIOZ      LOOP-2               Use hardware to detect end of loop

          ZALS      MIN-AMDF-L,Ø         Compare current AMDF value w/
          ADDH      MIN-AMDF-H,Ø           previous minimum
          SUB       AMDF-L,Ø
          SUB       AMDF-H,15
          BLZ       SAME
          ZALS      AMDF-L,Ø             If smaller, update minimum
          ADDH      AMDF-H,Ø
          SACL      MIN-AMDF-L,Ø
          SACH      MIN-AMDF-H,Ø
          ZALS      *,ARØ                Save present pitch value
          SACL      T,Ø
```

```
SAME    MAR     *+,ARØ
        ZALS    NUM-PITCHS,Ø
        SUB     ONE,Ø
        SACL    NUM-PITCH,Ø
        BGEZ    LOOP-1              Loop if not finished
```

APPENDIX III

PITCH PREDICTION COEFFICIENT -VERSION I

```
*
*  Compute the pitch prediction coefficient.  Assumes speech is stored in external
*  RAM access using TBLR
*

INIT    ZAC                             Initialize numerator and denominator
        SACL    NLM-L,Ø
        SACH    NUM-H,Ø
        SACL    DEN-L,Ø
        SACH    DEN-H,Ø
        LACK    #S-ADDR                 Initialize pointers to s[n] & s[n-T]
        SACL    S-PTR,Ø
        SUB     T,Ø
        SACL    ST-PTR,Ø
        LARK    ARØ, #N                 Initialize loop counter ARØ

LOOP    ZALS    S-PTR
        TBLR    S                       Read s[n]
        ADD     ONE, Ø
        SACL    S-PTR, Ø
        ZALS    ST-PTR, Ø
        TBLR    ST                      Read s[n-T]
        ADD     ONE
        SACL    ST-PTR
        ZALS    NUM-L, Ø
        ADDH    NUM-H, Ø
        LT      ST
        MPY     S
        APAC
        SACL    NUM-L,Ø                 Update numerator
        SACH    NUM-H, Ø
        ZALS    DEN-L,Ø
        ADDH    DEN-H,Ø
        MPY     ST
        APAC
        SACL    DEN-L,Ø                 Update denominator
        SACH    DEN-H, Ø
        MAR     *-, ARØ
        BNAZ    LOOP
        CALL    DIVIDE                  Perform divide in a subroutine
        ZACH    QUOTIENT
        SACH    ALPHA                   Store result
```

APPENDIX IV

PITCH PREDICTTION COEFFICIENT - VERSION II

```
*
*  Compute pitch predictor coefficient α.  Assume speech is stored in external RAM
*  accessed via the I/O ports using the IN instruction.

INIT    ZAC                                  Initialize numerator & denominator to
        SACL    NUM-L,Ø                        zero
        SACH    NUM-H,Ø
        SACL    DEN-L,Ø
        SACH    DEN-H,Ø
        LACK    #1                           Initialize external memory interface
        SACL    ACCESS-CTR-WD,Ø                for Mode 1 type memory interface
        CALL    EXT-MEM-INIT

LOOP    IN      S                            Input s[n] & s[n-T]
        IN      ST
        ZALS    NUM-L,Ø
        ADDH    NUM-H,Ø
        LT      ST
        MPY     S
        APAC
        SACL    NUM-L                        Update numerator
        SACH    NUM-H
        ZALS    DEN-L
        ADDH    DEN-H
        MPY     ST
        APAC
        SACL    DEN-L,Ø                      Update denominator
        SACH    DEN-H,Ø
        BIOZ    LOOP                         Detect end of loop in hardware
        CALL    DIVIDE
        ZALH    QUOTIENT, Ø                  Compute result and store it
        SACH    ALPHA,Ø
```

APPENDIX V

COMBINED 1ST RESIDUAL AND AUTOCORRELATION COMPUTATION

```
*
*  Computes 5 autocorrelation values along with the 1st residual direct
*  from the speech signal.  Assumes that speech is stored in external
*  RAM and that it is accessed using TBLR

INIT     LACK     #S-ADDR          Initialize pointers to speech
         SACL     S-PTR, Ø            data
         SUB      T,Ø
         SACL     ST-PTR,Ø
         LACK     #N               Initialize main loop counter
         SACL     COUNT, Ø
         LARK     ARØ, #R-ADDR
         LARK     AR1, #ORDER

LOOP-4   SACL     *+, ARØ          Initialize autocorrelation values
         MAR      *-, AR1             to zero
         BNAC     Loop-4

LOOP-1   ZALS     S-PTR,Ø
         TBLR     S                Read s[n]
         SUB      ONE,Ø
         SACL     S-PTR,Ø
         LT       ALPHA
         ZALS     ST-PTR,Ø
         TBLR     ST               Read s[n-T]
         SUB      ONE,Ø
         SACL     ST-PTR,Ø
         ZALS     S                Compute e1[n] = s[n]-   s[n-T]
         MPY      ST
         SPAC                      Place e1 [n] value on a First-in
         LARK     ARØ, #e-ADDR        first-out, stack which retains
         SACL     *, Ø, ARØ           five most previous residuals values
         LT       *, ARØ
         LACK     #ORDER           Set up loop to compute correlation
         SACL     COR-COUNT, Ø        values
         LARK     AR1, # -ADDR

                                   Update, RØ through R4
LOOP-2   ZALS     *+,AR1,Ø            values
         ADDH     *-,AR1,Ø
         MPY      *-,AR1,Ø
         APAC
         SACL     *+,AR1,Ø
         SACH     *+,AR1,Ø
         ZALS     COR-COUNT,Ø
         SUB      ONE
```

```
        SACL    COR-COUNT,Ø
        BNEZ    LOOP-2
        LARK    ARØ, #e4-ADDR       Reorder values in the residual FIFO
        LARK    AR1, #ORDER

Loop-3  DMDV    *-,ARØ,Ø
        MAR     *-,AR1
        BNAC    LOOP-3
        ZALS    COUNT,Ø               Update loop counter
        SUB     ONE,Ø
        SACL    COUNT,Ø
        BNEZ    LOOP-1                REDO LOOP
```

APPENDIX VI

LEROUX-GUEGUEN RECURSION FOR COMPUTING LPC PARCOR COEFFICIENTS

```
*
*
*
LPC-INIT    LACK    #4                  Set up pointers to transfer
            SACL    INIT-COUNTER,Ø      autocorrelation values to init recursion
            LARK    ARØ, #ADDR-RØ       Point to R[Ø]
            LARK    AR1, #ADDR-EØ       Point to e^n[Ø]

INIT-LP-1   ZALS    *+,ARØ              Auto correlations are double word
            SACH    *+,AR1                 (use upper word only)
            MAR     *+,ARØ,             (skip a word)
            ZALS    INIT-COUNTER
            SUB     ONE,Ø
            SACL    INIT-COUNTER
            BNEZ    INIT-LP-1

            LACK    #3                  Load the other three autocor. values
            SACL    INIT-COUNTER
            LARK    ARØ, #ADDR-R1       Aux registers pointed to data
            LARK    AR1, #ADDR-E-1

INIT-LP-2   ZALS    *+,ARØ
            SACL    *+,AR1,Ø
            MAR     *+,ARØ              skip word for double worded
            ZALS    INIT-COUNTER           autocorrelation
            SACL    INIT-COUNTER,Ø
            BNEZ    INIT-LP-2

INIT-PTRS   LARK    ARØ, #ADDR E-ARRAY  init pointer to e^(n)[-p+n+2]
            SAR     ARO, E-ARRAY-START
            LARK    ARØ, #ADDR-kØ
            SAR     ARØ, K-PTR
            LACK    #7                  init number of iterations
            SACL    NUM-INTERATIONS, Ø
            LARK    ARØ, #ADDR-ENØ      init pointer to e^(n)[Ø]
            SAR     ARØ, ENØ-PTR
            LARK    ARØ, #ADDR-EN1      init pointer to e^(n)[n+1]
            SAR     ARØ, EN1-PTR

LG-REC      LAR     ARØ,EN1-PTR
            ZALS    *+,ARØ
            SACL    NUMERATOR
            SAR     ARØ, ENT-PTR
            LAR     ARØ, FNØ-PTR
            ZALS    *-ARØ
            SACL    DENOMINATOR
            CALA    DIVIDE
```

```
            ZALS        QUOTIENT
            LAR         ARØ,K-PTR
            SACL        *,ARØ,Ø
            LT          *+,ARØ

            SAR         ARØ, K-PTR
            LARK        ARØ, #ADDR-E-END        Init pointers for from data
            LAR         AR1, E-ARRAY-START
            SAR         AR1, CROSS-PTR
            LARK        AR1, #ADDR-SCR-END      Init ptr to put cway data
            SAR         AR1, TO-PTR
            ZALS        NUM-ITERATIONS
            BEZ         DONE
            SACL        LG-COUNTER

LG LOOP     ZALS        *+,ARØ, Ø               e(n)[i]   Accumulator
            LAR         AR1, CROSS-PTR
            MPY         *+, AR1                 k[n]e(n)[n+1-i]
            SPAC
            SAR         AR1, CROSS-PTR
            LAR         AR1, TO-PTR
            SACL        *+, AR1, Ø
            SAR         AR1, TO-PTR
            ZALS        LG-COUNTER
            SUB         ONE
            SACL        LG-COUNTER,Ø
            BNEZ        LG-LOOP

            ZALS        NUM-ITERATIONS
            SACL        LG-COUNTER, Ø
            LARK        ARØ, #ADDR-E-END

TERLP       LARK
            BALS          *+, AR1, Ø
            SACL          *+,ARØ, Ø
            ZALS          LG-COUNTER
            SLB           ONE
            SACL          LG-COUNTER, Ø
            BNEZ          TFR-LP
```

```
ZALS    NUM-ITERATIONS
SUB     ONE, Ø
SACL    NUM-ITERATIONS, Ø
LAR     ARØ, E-ARRAY-START
MAR     *+, ARØ
SAR     ARØ, E-ARRAY-START
B       LG-REC
```

APPENDIX VII

PREDICTIVE QUANTIZER LOOP - VERSION I

```
*
*  Assumes speech data as well as the pitch predictor state space
*  is stored in external RAM.  The pitch predictor state space is kept
*  in a circular buffer which is accessed using TBLR and TBLW
*  instructions

INIT      ZALS      N, Ø
          SACL      COUNTER, Ø          Initialize Loop Counter
          ZALS      #S-ADDR, Ø          Initialize Pointer to Incoming speech
          SACL      S-PTR

LOOP-1    LARK      ARØ, #A-ADDR        ARØ points to predictor coefficients
          LARK      AR1, # E1-ADDR      AR1 points to spectral filter
          ZAC                             state space
          LT        *+, AR1             Compute spectral predictor
          MPY       *+, ARØ

LOOP-2    LTD       *+, AR1
          MPY       *+, ARØ
          BANZ      LOOP-2
          APAC
          SACH      Y, Ø                Y contains Spectral Prediction
          ZAC                           COMPUTE PITCH PREDICTION
          LT        ALPHA
          MPY       RT
          APAC
          SACH      X, Ø                X CONTAINS PITCH PREDICTION
          ZALS      S-PTR, Ø
          TBLR      S
          ADD       ONE, Ø
          SACL      S-PTR, Ø
          ZALS      S                   Subtract two predictions from
          SUB       X, Ø                  input speech
          SUB       Y, Ø
          BGEZ      DIFF-POS            Quantize residual, if neg.
          ZAC                             transmit zero
          SACL      D, Ø                Scale variance of quantized
          LT        Ø                     residual
          MPYK      MINUS-1
          APAC
          B         UPDATE

DIFF-POS  LACK      ONE                 If residual is positive, transmit
          SACL      D, Ø                  one
          ZALS      Q, Ø
```

```
UPDATE      ADD       Y, Ø          Compute spectral prediction filter
            SACH                      state variable
            ADD       X, Ø
            SACH      R, Ø          Compute Pitch Predictor State
                                      variable and store it

            ZALS      R-OUT-PTR,Ø   Compute pointer into circular buffer
            ADD       ONE,Ø              for storing data
            SACL      R-OUT-PTR,Ø
            SUB       R-BOTTOM, Ø
            BLZ       OUTPUT-R
            LACK      #ADDR-RBUF    If at end of buffer, point
            SACL                       back to beginning

OUTPUT-R    ZALS      R-OUT-PTR, Ø
            TBLW      R

            ZALS      R-IN-PTR, Ø   Compute pointer into circular
            ADD       ONE, Ø           buffer for retrieving data
            SACL      R-IN-PTR, Ø
            SUB       R-BOTTOM, Ø
            BLZ       INPUT-R
            LACK      #ADDR-RBUF
            SACL      R-IN-PTR, Ø

INPUT-R     ZALS      R-IN-PTR, Ø
            TBLR      RT

            ZALS      COUNTER, Ø
            SUB       ONE
            SACH      COUNTER, Ø
            BNEZ      LOOP-1
```

APPENDIX VIII

PREDICTIVE QUANTIZER LOOP - VERSION II

```
*
*  Assumes input speech and reconstructed speeech data to be stored
*  in external RAM.  The input speech is accessed using TBLR.  The
*  reconstructed speech is accessed using the I/O ports and the exter-
*  nal memory interface.  The serial quantized residual signal is packed
*  into 16-bit words.

INIT    ZALS    N,Ø                 Initialize loop counter
        SACL    COUNTER, Ø
        LACK    #S-ADDR,Ø           Initialize Pointer to input speech
        SACL    S-PTR,Ø               data
        LACK    #D-ADDR
        SACL    D-OUT-PTR,Ø
        ZALS    THREE
        SACL    ACCESS-CTR-WD,Ø     Initialize external memory
        CALL    EXT-MEM-INIT          interface for read/write mode
        LACK    #16
        SACL    COUNTER, Ø
        ZAC     BYTE
        SACL    RESIDUAL-BYTE,Ø

Loop-1  LARK    ARØ, #A-ADDR        Compute spectral prediction
        LARK    AR1,#E1-ADDR
        ZAC
        LT      *-,AR1
        MPY     *-,ARØ

Loop-2  LTD     *-,AR1
        MPY     *-,ARØ
        BANZ    LOOP-2
        APAC
        SACH    Y,0
        ZAC                         Compute pitch prediction
        LT      ALPHA
        MPY     RT
        APAC
        SACH    X,Ø
        ZALS    S-PTR,Ø
        TBLR    S-PTR               Read input speech sample
        ADD     ONE, Ø
        SACL    S-PTR
        ZALS    S
        SUB     X,Ø
        SUB     Y,Ø                 Compute residual and quantize
        BGEZ    DIFF-POS
```

```
            ZAC
            SACL    D,Ø             If quantized residual is negative
            LT      Q                  transmit zero
            MPYK    MINUS 1
            APAC
            B       UPDATE

DIFF-POS    LACK    #ONE            If residual is positive transmit one
            SACL    D,Ø
            ZALS    Q,Ø

UPDATE      ADD     Y,Ø             Combine quantized first residual
            SACH    E1, Ø
            ADD     X, Ø            Compute reconstructed speech and
                                       store it
            OUT     R, RAM-PORT-1
            IN      RT, RAM-PORT-2

            ZALS    BYTE-COUNTER,Ø  Pack residual bit stream
            SUB     ONE, Ø
            SACL    BYTE-COUNTER
            BLEZ    NEW-OUT-BYTE
            ZALS    RESIDUAL-BYTE, 1
            ADD     D,Ø
            B       GO-ON

NEW         ZALS    D-OUT-PTR,Ø
            TBLW    RESIDUAL-BYTE
            ADD     ONE, Ø
            SACL    D-OUT-PTR,Ø
            ZALS    D,Ø
            SACL    RESIDUAL-BYTE
            LACK    #16
            SACL    BYTE-COUNTER,Ø

GO-ON       ZALS    COUNTER,Ø
            SUB     ONE,Ø
            SACL    COUNTER,Ø
            BGEZ    LOOP-1
```

APPENDIX IX

RECEIVER LOOP

Assumes residual input is packed in 16-bit words in external RAM. It is accessed using TBLR. Synthesized speech is stored in a circular buffer in external RAM. This buffer is accessed via the external I/O Ports.

```
:T      LACK    #N              Initialize a loop counter
        SACL    COUNTER,Ø
        LACK    #D-IN-ADDR      Initialize pointer to input residual
        SACL    D-IN-PTR,Ø
        ZALS    FOUR
        SACL    ACCESS-CTR-WD,Ø Initialize external memory interface
        CALL    EXT-MEM-INIT       for Read/Write mode
        ZAC                     Initialize counter for parallel
        SACL    BYTE-COUNTER       to serial conversion of input residual

)P-1    ZALS    BYTE-COUNTER    Obtain quantized input residual
        BNEZ    SHIFT              from 16-bit residual word
        ZALS    D-IN-PTR
        TBLR    RESIDUAL-BYTE,Ø
        ADD     ONE,Ø
        SACL    D-IN-PTR
        LACK    #16
        SACL    BYTE-COUNTER,Ø

:FT     ZALS    RESIDUAL-BYTE,I Current residual value is high order
        SACL    RESIDUAL-BYTE,Ø    bit of residual-byte
        SACH    P,Ø
        ZALS    BYTE-COUNTER,Ø
        SUB     ONE,Ø
        SACL    BYTE-COUNTER,Ø

        LT      Q               Scale variance of residual
        MPY     D
        ZAC
        APAC
        ADD     Y,Ø             Add in spectral prediction
        SACH    E,Ø                put result on spectral pred.
        ADD     X,Ø                filter state space
        SACH    S-HAT,Ø         Store reconstructed speech
        OUT     S-HAT, RAM-PORT-1  in circular buffer
        LARK    ARØ,#A-ADDR     Compute next spectral prediction
        LARK    AR1, #E-ADDR       value
        LT      *-,AR1
        MPY     *-,ARØ
        ZAC
```

```
LTD             #-,AR1
MPY             *-,AR0
BNAC            LOOP-2
SACH            Y,0

IN              ST-HAT              Compute next pitch prediction
LT              ALPHA
MPY             ST-HAT
ZAC
APAC
SACH            X,0
ZALS            COUNTER,0
SUB             ONE,0
SACL            COUNTER,0
BNEZ            LOOP-1
```

APPENDIX X

ADDA A/D-D/A SERVICE ROUTINE INVOKED BY

INTERRUPT FROM A/D CLOCK

```
*
*  A/D portion
*
AD          IN          SN, ADC             Input speech from ADC Register
            ZALS        SN, Ø               Pre-emphasis
            LT          OLDSN
            MPYK        PRE-FAC
            SPAC
            SACL        TEMP,Ø              Store preemphasized speech temp.
            ZALS        S-IN-PTR,Ø          Load pointer to input speech buffer
            TBLW        TEMP                Write out preemphasized speech in buffer
            ADD         ONE,Ø                  increment pointer
            SACL        S-IN-PTR,Ø
            ZALS        SN,Ø                delay s[n]
            SACL        OLDSN
*
*  D/A Portion
*           ZALS        S-OUT-PTR-1,Ø       Retrieve pntr to output speech buffer
            TBLR        YN                  Real in processed speech sample
            ADD         ONE,Ø               Increment pointer
            SACL        S-OUT-PTR-1,Ø       Re-store pointer
            ZALS        YN                  Do De-emphasis
            LT          OLD SHATN
            MPYK        PRE-FAC
            APAC
            SACL        OLD SHATN,Ø         Delay output speech sample
            OUT         OLD-SHATN,DAC       Output speech sample

*
*  Check for end of buffer.  If the end, switch speech buffers.
*  This is done by switching pointers.

            ZALS        S-PTR-1,Ø           Check for end of Data
            SUB         S-OUT-END,Ø
            BGZ         DONE
            ZALS        S-OUT-PTR-1,Ø       Toggle bit 8, switching from page
            XOR         H'100               5 to 6 (and vice versa)
            SACL        S-OUT-PTR-1,Ø
            ZALS        S-OUT-PTR-Z,Ø
            XOR         H'100
            SACL        S-AT-PTR-2

DONE        RET                             Return from interrupt
```

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-84-276	2. GOVT ACCESSION NO. AD-A152461	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* The Design of An Adaptive Predictive Coder Using a Single-Chip Digital Signal Processor		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER Technical Report 679
7. AUTHOR(s) Mark A. Randolph		8. CONTRACT OR GRANT NUMBER(s) F19628-85-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173-0073		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Element Nos. 33401F and 63735F
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Systems Command, USAF Andrews AFB Washington, DC 20331		12. REPORT DATE 11 January 1985 13. NUMBER OF PAGES 72
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* Electronic Systems Division Hanscom AFB, MA 01731		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

vocoders
speech compression
adaptive predictive coding
digital signal processing microcomputers
speech processor architectures
special purpose processor

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A speech coding processor architecture design study has been performed in which the Texas Instruments TMS32010 has been selected from among three commercially available digital signal processing integrated circuits and evaluated in an implementation study of real-time Adaptive Predictive Coding (APC). The TMS32010 has been compared with the AT&T Bell Laboratories DSP I and Nippon Electric Co. PD7720 and was found to be most suitable for a single chip implementation of APC. A preliminary system design based on the TMS32010 has been performed, and several of the hardware and software design issues are discussed. Particular attention was paid to the design of an external memory controller which permits rapid sequential access of external RAM. As a result, it has been determined that a compact hardware implementation of the APC algorithm is feasible based on the TMS32010.

DD FORM 1 Jan 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

ORIGINATOR - SUPPLIED KEY WORDS INCLUDE:

END

FILMED

5-85

DTIC